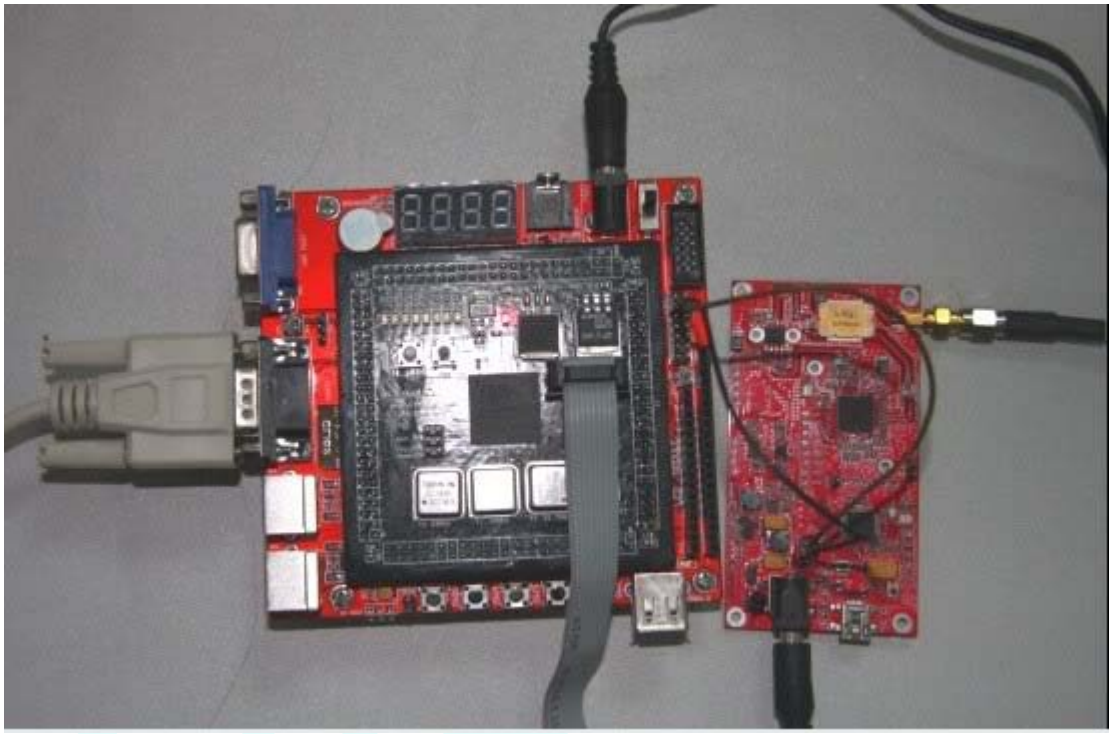
The hardware includes the following:

1.TTL UART UHF RFID reader module

2.PPGA Board

3.UHF RFID antenna

The following is a system picture:



The jump wire connection is as follows:

1. FPGA GPIO pin 13 -> UHF RFID GND.
2. FPGA GPIO pin 3 -> UHF RFID TX.
3. FPGA GPIO pin 4-> UHF RFID RX.

FPGA codes are as following：

1.Top module:

```
module my_uart_top(clk,rst_n,rx,tx,key_input,txpc);
input clk; // 50MHz major clock
input rst_n;    //low level reset signal
input rx;     // FPGA receives data signal
output tx;    // FPGA sends data signal
input key_input;    //FPGA button input
output txpc;    //FPGA's RS232 output
wire bps_start;     //after receives the data, Band rate start signal position.
wire clk_bps;        // clk_bps high level is the middle sampling point of the data receiving or sending.
wire[7:0] rx_data;     //data receiving register. Save until next data comes.
wire rx_int;    //stop signal of data receiving. Keep high level during data receiving.
```

```
//----------------------------------------------------
speed_select    speed_select(    .clk(clk), //Band rate chooses module. Receiving and sending module reuses. It doesn't support full-duplex communication.
                                 .rst_n(rst_n),
                                 .bps_start(bps_start),
                                 .clk_bps(clk_bps)
                                 );

my_uart_rx      my_uart_rx(      .clk(clk), //data receiving module
                                 .rst_n(rst_n),
                                 .rx(rx),
                                 .clk_bps(clk_bps),
                                 .bps_start(bps_start),
                                 .rx_data(rx_data),
                                 .rx_int(rx_int),
                .txpc(txpc)
                                 );

my_uart_tx      my_uart_tx(      .clk(clk), //data sending module
                                 .rst_n(rst_n),
                                 .tx(tx),
                .key_input(key_input)
                                 );
endmodule
```

2.Band rate chooses module（here is synchronous with RFID, and is 115200bps）

```
module speed_select(clk,rst_n,bps_start,clk_bps);
input clk; // 50MHz major clock
input rst_n;   //low level reset signal
input bps_start;   // after receives the data, Band rate start signal position
output clk_bps;    // clk_bps high level is the middle sampling point of the data receiving or sending.
parameter       bps9600      = 5207,      //Band rate is 9600bps
      bps19200     = 2603,      // Band rate is 19200bps
      bps38400     = 1301,      // Band rate is 38400bps
      bps57600     = 867,       // Band rate is 57600bps
      bps115200    = 433;       // Band rate is 115200bps
parameter       bps9600_2 = 2603,
      bps19200_2 = 1301,
      bps38400_2 = 650,
      bps57600_2 = 433,
      bps115200_2 = 216;
reg[12:0] bps_para;          // maximum value of frequency division count
```

```
reg[12:0] bps_para_2;       //half of the frequency division count
reg[12:0] cnt;              //frequency division count
reg clk_bps_r;              //Band rate clock register

//-----------------------------------------------------------
   reg[2:0] uart_ctrl;    // uart Band rate chooses register
//-----------------------------------------------------------
 always @ (posedge clk or negedge rst_n) begin
     if(!rst_n) begin
       uart_ctrl <= 3'd4;     //Default Band rate is 115200bps
         end
     else begin
            case (uart_ctrl)    //Band rate set
               3'd0:    begin
                         bps_para <= bps9600;
                         bps_para_2 <= bps9600_2;
                         end
               3'd1:    begin
                         bps_para <= bps19200;
                         bps_para_2 <= bps19200_2;
                         end
               3'd2:    begin
                         bps_para <= bps38400;
                         bps_para_2 <= bps38400_2;
                         end
               3'd3:    begin
                         bps_para <= bps57600;
                         bps_para_2 <= bps57600_2;
                         end
               3'd4:    begin
                         bps_para <= bps115200;
                         bps_para_2 <= bps115200_2;
                         end
              default: ;
               endcase
         end
end

always @ (posedge clk or negedge rst_n)
     if(!rst_n) cnt <= 13'd0;
     else if(cnt<bps_para && bps_start) cnt <= cnt+1'b1;    //Band rate clock count starts
     else cnt <= 13'd0;
```

```
always @ (posedge clk or negedge rst_n)
    if(!rst_n) clk_bps_r <= 1'b0;
    else if(cnt==bps_para_2 && bps_start) clk_bps_r <= 1'b1;    // clk_bps_r high level is the middle sampling point of the data receiving or sending.
    else clk_bps_r <= 1'b0;
assign clk_bps = clk_bps_r;
endmodule
```

3.Data module sending

```
module   my_uart_tx(clk, rst_n, tx, key_input);
   input     clk,rst_n;
 input     key_input;    //button input
 output    tx;       //serial data sending terminal

 //*************************inner reg*****************************//
 reg[15:0]   div_reg;     //frequency division counter. Frequency division 分频 value is decided by Band rate. After division, you can get the clock with the frequency 8 times of Band rate.
 reg[2:0]    div8_tras_reg;    //count value of this register is corresponding to the number of time slots at the data sent.
 reg[3:0]   state_tras;     //sending status register
 reg      clkbaud_tras;    //take Band rate as frequency to send enable signal.
 reg      clkbaud8x;    //this clock has the frequency that is 8 times of Band rate. It is used to separate one bit clock circle (sending/receiving) into 8 time slots)

 reg      trasstart;     //start to send signal
 reg      txd_reg;     //sending register
 reg[7:0]   txd_buf;      //sending data buffer
 reg[1:0]   send_state;     //each button click send “430301” to PC. This is the sending status register.
 reg[22:0]   cnt_delay;    //debounce delay counter
 reg      start_delaycnt;       //delay counting start mark
 reg      key_entry1, key_entry2; //button pressing determined mark
 parameter    div_par = 8'h1B;       //frequency division parameter. Its value is calculated from corresponding Band rate. The clock frequency upon this parameter frequency division is 8 times of Band rate. Here, the corresponding Band rate is 115200. The clock frequency after division is 115200*8   (CLK   50M)
 //*************************************************************//
 assign     tx = txd_reg;

 always@(posedge clk or negedge rst_n)
 begin
  if(!rst_n)begin
```

```
    cnt_delay <= 0;
    start_delaycnt <= 0;
          end
  else if(start_delaycnt) begin
    if(cnt_delay != 23'd8000000) begin
      cnt_delay <= cnt_delay + 1;
      end
    else begin
      cnt_delay <= 0;
      start_delaycnt <= 0;
        end
    end
  else   begin
    if(!key_input && cnt_delay == 0)
       start_delaycnt <= 1;
      end
end

//*********************************************************//
always@(posedge clk or negedge rst_n)
begin
  if(!rst_n)
    key_entry1 <= 0;
  else   begin
    if(key_entry2)
      key_entry1 <= 0;
    else if(cnt_delay == 23'd8000000) begin
      if(!key_input)
        key_entry1 <= 1;
      end
    end
end

//*********************************************************//
always@(posedge clk or negedge rst_n)
begin
  if(!rst_n)
    div_reg <= 0;
  else                begin
    if(div_reg == div_par - 1)
      div_reg <= 0;
    else
```

```
      div_reg <= div_reg + 1; end
  end
  always@(posedge clk or negedge rst_n)     //frequency divided to get 8 times Band rate clock.
  begin
    if(!rst_n)
      clkbaud8x <= 0;
    else if(div_reg == div_par - 1)
      clkbaud8x <= ~clkbaud8x;
  end

  always@(posedge clkbaud8x or negedge rst_n)
  begin
    if(!rst_n)
      div8_tras_reg <= 0;
    else if(trasstart)
      div8_tras_reg <= div8_tras_reg + 1; //after sending started, under the 8 times Band rate, the time slot add 1 cycle.
  end
  //************************************************************//
  always@(div8_tras_reg)
  begin
    if(div8_tras_reg == 7)
      clkbaud_tras = 1;     //at the seventh time slot, sending enable signal is valid. Send data out.
    else
      clkbaud_tras = 0;
  end

  //************************************************************//
  always@(posedge clkbaud8x or negedge rst_n)
  begin
    if(!rst_n) begin
      txd_reg <= 1;
      trasstart <= 0;
      txd_buf <= 0;
      state_tras <= 0;
      send_state <= 0;
      key_entry2 <= 0;
      end
    else begin
      if(!key_entry2) begin
```

```
    if(key_entry1) begin
     key_entry2 <= 1;
     txd_buf <= 8'h43;
     end
    end
   else begin
    case(state_tras)
     4'b0000:      begin    //sending start bit          if(!trasstart && send_state < 3)
       trasstart <= 1;
      else if(send_state < 3)          begin
       if(clkbaud_tras)          begin
        txd_reg <= 0;
        state_tras <= state_tras + 1;end end
      else                  begin
       key_entry2 <= 0;
                                  send_state <= 0;
        state_tras <= 0;           end end
     4'b0001:                begin //send the first bit
      if(clkbaud_tras)          begin
       txd_reg <= txd_buf[0];
       txd_buf[6:0] <= txd_buf[7:1];
       state_tras <= state_tras + 1;    end end
     4'b0010:                begin //send the second bit
      if(clkbaud_tras)          begin
       txd_reg <= txd_buf[0];
       txd_buf[6:0] <= txd_buf[7:1];
       state_tras <= state_tras + 1;    end end
      4'b0011:               begin //send the third bit
      if(clkbaud_tras)          begin
       txd_reg <= txd_buf[0];
       txd_buf[6:0] <= txd_buf[7:1];
       state_tras <= state_tras + 1;    end end
     4'b0100:                begin //send the fourth bit
      if(clkbaud_tras)          begin
       txd_reg <= txd_buf[0];
       txd_buf[6:0] <= txd_buf[7:1];
       state_tras <= state_tras + 1;    end end
     4'b0101:                begin //send the fifth bit
      if(clkbaud_tras)          begin
       txd_reg <= txd_buf[0];
       txd_buf[6:0] <= txd_buf[7:1];
       state_tras <= state_tras + 1;    end end
```

```
4'b0110:              begin //send the sixth bit
 if(clkbaud_tras)          begin
  txd_reg <= txd_buf[0];
  txd_buf[6:0] <= txd_buf[7:1];
  state_tras <= state_tras + 1;   end end
4'b0111:              begin //send the seventh bit
 if(clkbaud_tras)          begin
  txd_reg <= txd_buf[0];
  txd_buf[6:0] <= txd_buf[7:1];
  state_tras <= state_tras + 1;   end end
4'b1000:              begin //send the eighth bit
 if(clkbaud_tras)          begin
  txd_reg<=txd_buf[0];
  txd_buf[6:0]<=txd_buf[7:1];
  state_tras<=state_tras+1;
  end
 end
4'b1001: begin //send stop bit
 if(clkbaud_tras) begin
  txd_reg<=1;
  txd_buf<=8'h55;
  state_tras<=state_tras+1;
  end
 end
4'b1111:begin
 if(clkbaud_tras) begin
  state_tras<=state_tras+1;
  send_state<=send_state+1;
  trasstart<=0;
  case(send_state)
   3'b00:
    txd_buf<=8'h03;
   3'b01:
    txd_buf<=8'h01;
   default:
    txd_buf<=0;
   endcase
  end
 end
default: begin
 if(clkbaud_tras) begin
  state_tras<=state_tras+1;
```

```
          trasstart<=1;
          end
         end
        endcase
      end
    end
  end
endmodule
```

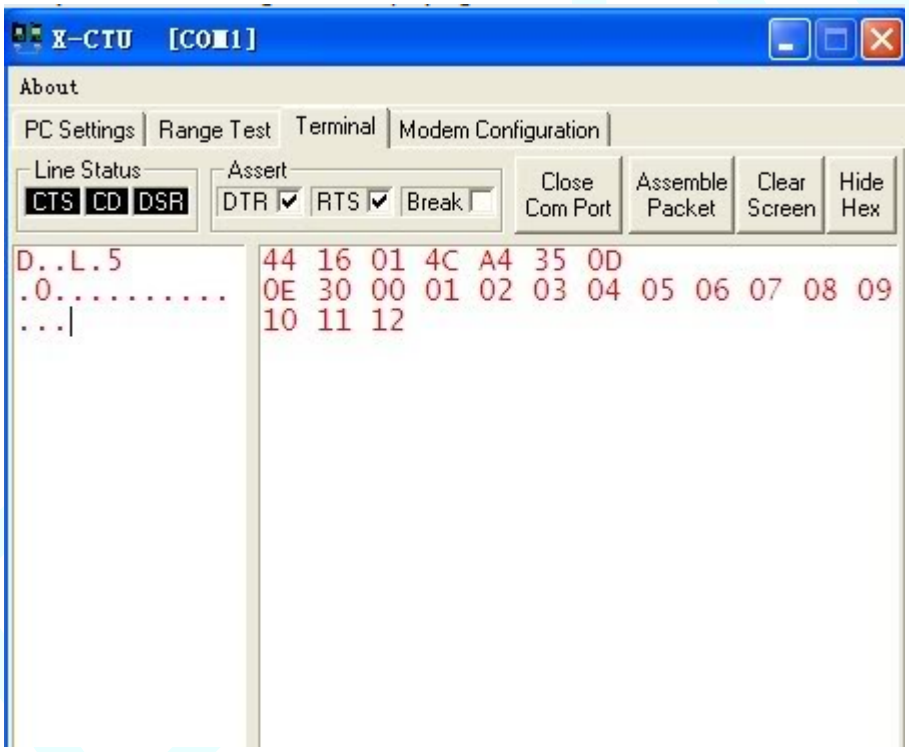
4. Data receiving module：

```
module my_uart_rx(clk,rst_n,rx,clk_bps,bps_start,rx_data,rx_int,txpc);
input clk;       // 50MHz major clock
input rst_n;    //low level reset signal
input rx;       // FPGA receiving data signal
input clk_bps;       // clk_bps high level is the middle sampling point of the data receiving or
sending.
output bps_start;    //after receive the data, Band rate clock start signal reset.
output[7:0] rx_data; //receive data register, keep until next data comes.
output rx_int;       //receive data stop signal. During this period, the level is high.
output txpc;       //RFID send data to PC through RS232
//---------------------------------------------------------------
reg rx0,rx1,rx2; //data receiving register used for filtering
wire neg_rx;     //means the falling of the data receiving line.
always @ (posedge clk or negedge rst_n) begin
      if(!rst_n) begin
            rx0 <= 1'b1;
            rx1 <= 1'b1;
            rx2 <= 1'b1;
         end
      else begin
            rx0 <= rx;
            rx1 <= rx0;
            rx2 <= rx1;
         end
end
assign neg_rx = rx2 & ~rx1; //after receive falling, neg_rx set one clock cycle high.
 //---------------------------------------------------------------
reg bps_start_r;
reg[3:0]     num;     //shift count
reg rx_int;     //receive data stop signal. During this period, the level is high.
 always @ (posedge clk or negedge rst_n) begin
      if(!rst_n) begin
            bps_start_r <= 1'bz;
```

```
            rx_int <= 1'b0;
        end
    else if(neg_rx) begin
            bps_start_r <= 1'b1; //start to receive data
            rx_int <= 1'b1;     // enable receiving data stop signal
            end
    else if(num==4'd10) begin
            bps_start_r <= 1'bz; //data receiving is finished
            rx_int <= 1'b0;       //close receiving data stop signal.
        end
end
assign bps_start = bps_start_r;

//-------------------------------------------------------------------
reg[7:0] rx_data_r;   //data receiving register. Keep until next data comes.
//-------------------------------------------------------------------

reg[7:0]    rx_temp_data; //current data receiving register
reg rx_data_shift;     //data shift mark
 always @ (posedge clk or negedge rst_n) begin
    if(!rst_n)
        begin
            rx_data_shift <= 1'b0;
            rx_temp_data <= 8'd0;
            num <= 4'd0;
            rx_data_r <= 8'd0;
        end
    else if(rx_int) begin       //receive data processing.
        if(clk_bps) begin //read and save data. Received data is the start bit. It is 8 bits data and one end bit.
                rx_data_shift <= 1'b1;
                num <= num+1'b1;
                if(num<=4'd8) rx_temp_data[7] <= rx;       //latch 9bit（1bit start byte，8bit data）
            end
        else if(rx_data_shift)
            begin       //data shift processing
                rx_data_shift <= 1'b0;
                if(num<=4'd8) rx_temp_data <= rx_temp_data >> 1'b1;   //data shift 8 times. Remove the 1bit start byte. When the 8bit left, start to receive data.
                else if(num==4'd10)
                    begin
```

```
                    num <= 4'd0;    //End after receive STOP bit, and num clear
                    rx_data_r <= rx_temp_data;    //latch data in data register rx_data
                end
            end
        end
    end
assign rx_data = rx_data_r;
assign txpc = rx; //send received data to PC. Display on PC serial debugging tools.

endmodule
```

Each time press the switch, key_input has low level. FAGA sends the set data 430301 which is the command for RFID to read label. After RFID read the label, there is a return value which sent back to FPGA by tx of RFID's uart. And then, FAGA sends it to PC by RS232. Following is the displaying result:



FPGA sends the data returned by RFID to PC and from the serial debugging software you can see there is one tag be found.